

# Static compilation of Julia packages for integration with existing HEP codebases: a case study with JetReconstruction.jl

---


Mateusz Fila (CERN)

Graeme A Stewart (CERN)

ACAT 2025, 08.09.2025

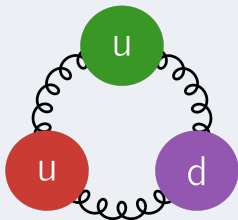





 [Julia website](https://julialang.org)



Modern general-purpose programming language oriented for high-performance and scientific computing but with composability and flexibility in mind.

- MIT License, invented in 2012
- C-like performance, Python-like flexibility
- Just-In-Time (JIT) compilation, garbage collector (GC), ML and data science packages, automatic differentiation (AD) notebooks, GPU support, Python/C++ wrappers, ...
- Strong contender for the future HEP computing language

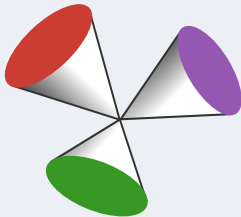



 [JuliaHEP website](#)  [Organization](#)

Growing collection of open-source, HEP specific packages for Julia with seamless integration with the rest of the Julia ecosystem.

- Active community, channel on Julia's slack, annual workshop
- Provides Julia bindings and native Julia re-implementations of many HEP libraries:
  -  [UnROOT.jl](#)
  -  [Geant4.jl](#)
  -  [EDM4hep.jl](#)
  -  [JetReconstruction.jl](#)
  - ...

# JetReconstruction.jl



 [JuliaHEP/JetReconstruction.jl](https://github.com/JuliaHEP/JetReconstruction.jl)

Native Julia implementation of sequential jet clustering algorithms based on FastJet.

- MIT License, community-driven development at GitHub.
- Started as a pilot project to investigate Julia's performance and ergonomics for HEP.
- The first stable release 1.0 coming soon.
- Available in Julia registry since summer 2024:

```
pkg> add JetReconstruction
```

# JetReconstruction features

## Clustering algorithms:

- $pp$  algorithms:  
Anti-kT, Cambridge/Aachen, kT,  
generalised kT  
Implementation strategies:  
N2Tiled, N2Plain, Best
- $e^+e^-$  algorithms:  
Durham, generalised kT

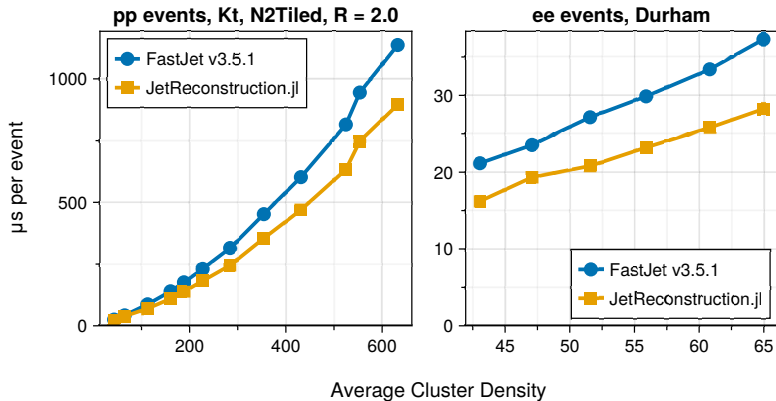
## Extras:

- Mass drop tagger.
- Pileup mitigation with SoftKiller.
- Lund Jet Plane implementation.
- Native visualizations with popular Julia plotting packages.
- Integration with EDM4hep types through EDM4hep.jl.

New features are in development, e.g., the Valencia algorithm.

# JetReconstruction.jl performance


- JetReconstruction consistently challenges or beats FastJet in performance benchmarks!



AlmaLinux 9, AMD Ryzen 7 5700G

# JetReconstruction.jl performance secrets

JetReconstruction achieves the performance by:

- Writing idiomatic Julia code and avoiding common pitfalls.
- Aggressive optimization of certain hot loop with  [JuliaSIMD/LoopVectorization.jl](https://github.com/JuliaSIMD/LoopVectorization.jl) (`@turbo` macro).

```
using LoopVectorization

fast_findmin(dij, n) = begin
    best = 1
    @inbounds dij_min = dij[1]
    @turbo for here in 2:n
        newmin = dij[here] < dij_min
        best = newmin ? here : best
        dij_min = newmin ? dij[here] : dij_min
    end
    dij_min, best
end
```

# JetReconstruction.jl use-cases

Current implementation of JetReconstruction is ideal for end-user analysis:

- Take events reconstructed in experiments' C++ frameworks.
- Conduct final analysis in Julia.

Performant C++ bindings may enable adoption by existing frameworks as a standalone algorithm:

- Target application is trigger farm for a HEP experiment.
  - Latency-sensitive — cannot afford any unexpected compilations.



# Directly embedding Julia in C++

- Uses Julia C-API.
- Fragile conversions and symbol lookup.
- Julia code is still JIT-compiled at runtime — calls into a complete Julia runtime stack.
- Needs to separately manage Julia environment.

Overall, not a good choice for our use-case.


```
#include <julia.h>
JULIA_DEFINE_FAST_TLS
int main() {
    jl_init();

    jl_function_t *func =
        jl_get_function(jl_base_module, "sqrt");
    jl_value_t *arg = jl_box_float64(2.0);
    jl_value_t *ret = jl_call1(func, arg);
    double value = jl_unbox_float64(ret);

    jl_atexit_hook(0);
    return 0;
}
```

# Julia AOT-compilation

Ahead-Of-Time (AOT) compilation of Julia code can produce standalone binaries:

- Requires writing C-bindings for functions (`@ccallable` macro) using only concrete C-compatible types.
- Significantly reduces or completely avoids compilation at runtime.
- We have investigated compilers:
  -  [JuliaLang/PackageCompiler.jl](https://github.com/JuliaLang/PackageCompiler.jl)
  - `juliac` from upcoming Julia v1.12.


Define in Julia and compile:

```
Base.@ccallable  
my_sqrt(x::Cdouble)::Cdouble  
= sqrt(x)
```

Link and use from C:

```
double my_sqrt(double);
```

# JetReconstruction bindings

- PR  [JuliaHEP/JetReconstruction.jl#88](https://github.com/JuliaHEP/JetReconstruction.jl/pull/88)
- Experimental C-interface and support for AOT-compilation.
- Started with *pp* reconstruction only.
- Distributed as shared library, header file and cmake config file.

# JetReconstruction C-interface

Lessons learned from writing the C-interface:

- Convenient data structures in Julia do not always map cleanly to C.
  - e.g. a struct with a vector field (which we have).
- Using mutable structures in the interface results in extra copying.
  - We made structures used in the interface immutable to avoid this.
- Returning Julia allocated memory to C is awkward due to garbage collector.
  - Our current algorithm design does this and changing that would be very intrusive.

Conclusion: Julia code should ideally be designed for this from the outset, it is not so easy to do this post-facto.

# AOT-compilation backends

PackageCompiler and juliac both work, but ...

- PackageCompiler needs a pre-compilation file that executes some code-paths. juliac tries to statically analyse the code to see what is reachable. Currently, both might not entirely eliminate JIT-compilation.
- PackageCompiler produces a complex set of libraries, which are difficult to deploy.
- PackageCompiler's performance with LoopVectorization is extremely poor. Otherwise no performance differences were observed.
- PackageCompiler has been broken for upcoming Julia v1.12 since beta3.

# AOT-compilation results

Code	Compiler	Compilation time	Binary size
JetReconstruction	juliac	~ 120 s	~ 300 MB
	PackageCompiler	~ 500 s	~ 300 MB
FastJet 3.5.1	gcc 14.2	~ 40 s ~ 20 s (parallel)	8 MB

AMD Ryzen 7 5700G

Overall, juliac seems to be a significant improvement over PackageCompiler.

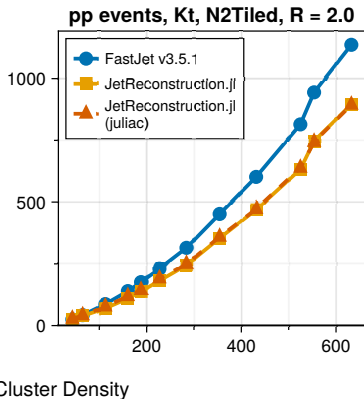
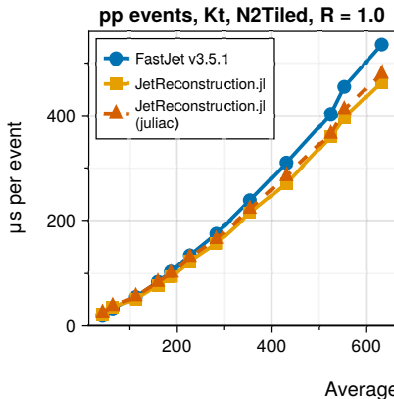
# juliac --trim=safe

juliac --trim=safe can further reduce sizes of produced binaries for codes using a sub-set of the language:

- For JetReconstruction this would mean significant changes including:
  - Removal of error handling with try-catch statements.
  - Removing logging printouts.
  - Dropping LoopVectorization (no querying CPU-features allowed).
  - Fixing syntax and various minor type instabilities.
- Even after these adjustments, two errors remain that prevent a trimmed build.
- Upcoming Julia patches are expected to ease the current constraints.

# JetReconstruction bindings performance

- Only slightly less performant than original JIT-compiled JetReconstruction.
- Some residual JIT-compilation still present with juliac, affects the first call. Expected to improve in future juliac patches.



AlmaLinux 9, AMD Ryzen 7 5700G




# Summary

- JetReconstruction.jl is a Julia package for jet clustering, already faster than FastJet in benchmarks.
- Ideal for end-user analysis in Julia. To enable adaption by existing C++ frameworks, we explored adding C-bindings.
- This required Ahead-Of-Time compilation, making JetReconstruction one of the first cases of compiling package with non-trivial interface using Julia's upcoming static compiler.
- We achieved successful builds, but the process revealed many quirks and limitations compared to the JIT-compiled version.
- We don't yet have a solid production solution.

# Static compilation of Julia packages for integration with existing HEP codebases: a case study with JetReconstruction.jl

 JuliaHEP/JetReconstruction.jl

- Mateusz Fila, *CERN*  
 [mateusz.jakub.fila@cern.ch](mailto:mateusz.jakub.fila@cern.ch)
- Graeme A Stewart, *CERN*

The work has been supported by the CERN Strategic Programme on Technologies for Future Experiments. <https://ep-rnd.web.cern.ch/>

This work has been partially funded by the Eric & Wendy Schmidt Fund for Strategic Innovation through the CERN Next Generation Triggers project under grant agreement number SIF-2023-004.

